

ADF Code Corner

98. How-to use multi select components in table filters



twitter.com/adfcodecorner

Abstract:

ADF Code Corner sample #16, I explains ADF Faces table filters can be customized, for example to show a `af:selectOneChoice` component instead of an input text field for users to define the filter criteria. 1 ½ years later, a question got posted on OTN asking how to make this sample work with a select many component. Here's the sample and documentation for how to build it.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
28-FEB-2012

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: <http://forums.oracle.com/forums/forum.jspa?forumID=83>

Important – Please Read!

18. September 2013 - Bug 16715661 - FILTER CAN'T BE REMOVED WHEN MULTISELECT FILTER IS USED ON ADF TABLE has been filed against the sample explained in this article.

According to the bug, when this multiple valued filter is used in conjunction with other simple filter criteria it works fine. If you try to remove or change the simple value filter, the simple filter value is restored and the query is executed again using the initial value.

I can reproduce the issue reported in the bug but don't see how to solve the issue from the query listener. A suggestion in the bug description is to clear the view criteria everytime before setting the new criteria map values. This ensures that the old VC is removed and a fresh VC is applied. I did not try this suggestion but think that the risk would be that the other simple filter criteria values are lost.

Until I find time to further investigate this issue I treat this issue as a limitation for using it with 11g R1 and 11g R2. If I find a solution, I'll update this article and remove this section.



Note that the problem does not seem to reproduce in JDeveloper 12c. Thus I like to narrow the scope of this article for it to be used only in JDeveloper 12c and newer release environments.

Note that JDeveloper 12c does not contain a notifyable fix for the issue that could be backported to earlier releases.

Sorry if this causes inconveniences.



Introduction

When you run the sample, the table filter show as if they are not customized.

View   Detach



DepartmentId	EmployeeId	FirstName
50	198	Donald
50	199	Douglas
10	200	Jennifer
20	201	Michael
20	202	Pat
40	203	Susan
70	204	Hermann
110	205	Shelley

However, clicking into the DepartmentId filter field immediately displays the select many choice component for the user to select filter values from

View   Detach

Marketing;Purchasin	EmployeeId	FirstName
		Donald
		Douglas
		Jennifer
		Michael
		Pat
		Susan
		Hermann
		Shelley
		William
		Stevensss
		Neena
		Lex
		Alexander
		Bruce
		David
		Valli
		Diana

As with the default table filter, hitting the Enter key executes the query based on the selected filter criteria

View   Detach

Marketing;Purchasin	EmployeeId	First
20	201	Mic
20	202	Pa'
30	114	De
30	115	Ale
30	116	Sh
30	117	Sig
30	118	Gu
30	119	Ka
60	103	Ale
60	104	Re

Note in the image above how the user selected query filter preserves its value.

Customize an ADF Faces table filter with an `af:selectManyChoice` component

In the ADF Code Corner article #16, the customization of an ADF Faces table filter with a select one choice component is explained in detail.

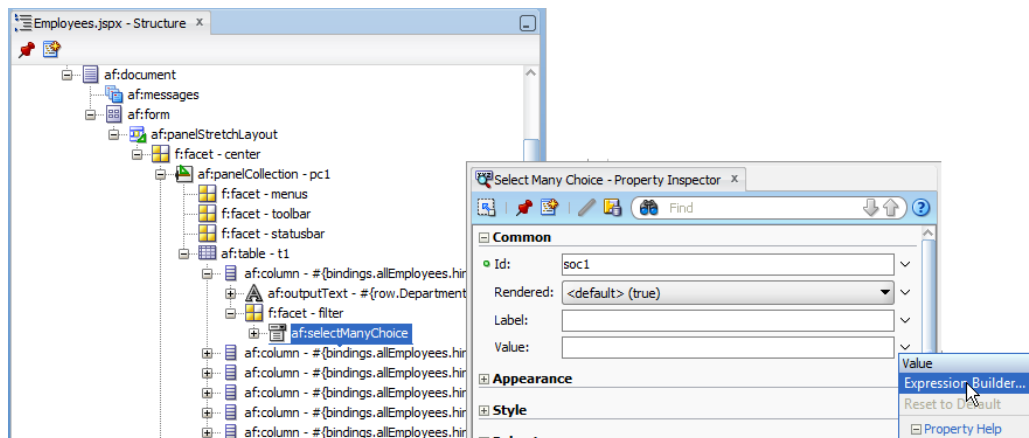
<http://www.oracle.com/technetwork/developer-tools/adf/learnmore/16-custom-table-filter-169145.pdf>

The steps listed in the following, explain how to add an `af:selectManyChoice` component to filter a column in an ADF Faces table

To customize the table filter, select the **“filter”** facet of an `af:column` component and choose “Insert inside filter” from the right mouse context menu to add a select many choice component. Use the Structure Window in Oracle JDeveloper for this

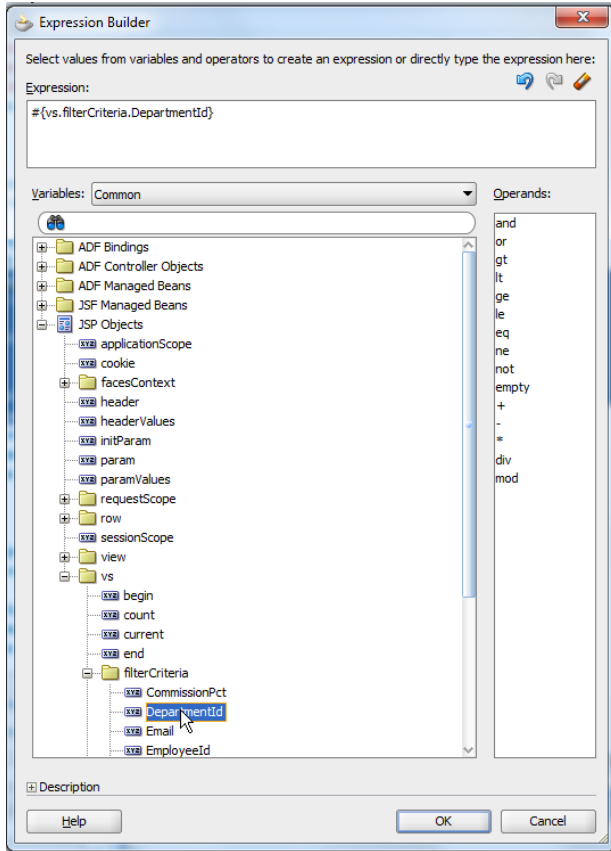
You can finish the select many choice dialog without a configuration, except for clearing the label field. All configurations to make the select one choice work as a filter component will be done manually in the following.

The `af:selectManyChoice` “value” attribute writes the selected list value as the filter criteria to the filter binding. To do so, select the “value” property in the Property Inspector and choose “Expression Builder” from the context menu as shown in the image below.



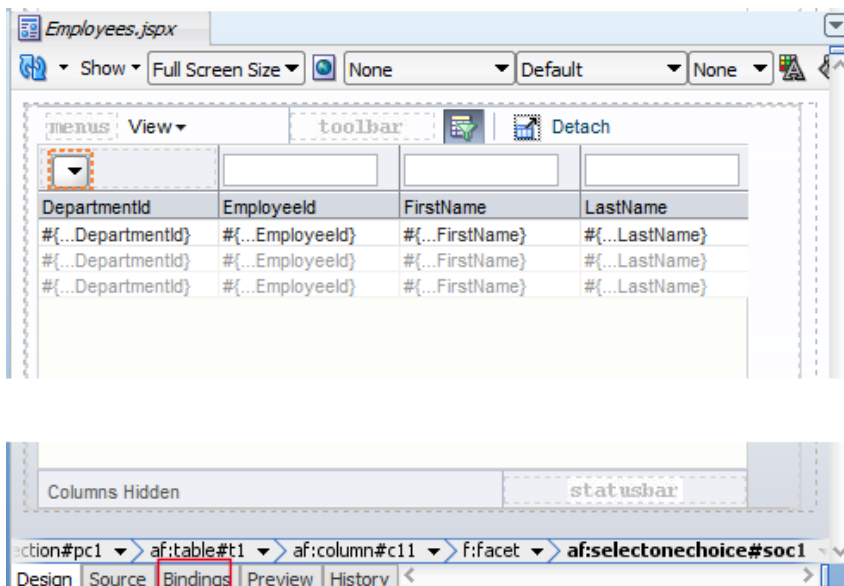
The filter criteria for a column is set using an EL string like `{vs.filterCriteria.<attribute name>}`.

To discover the attributes that are available to filter for a table, in the Expression Builder in Oracle JDeveloper, expand the JSP Objects node to access the “vs” node. Remember that the “vs” note is set as an attribute value on the table. So if the name differs, then you see a different value entry in the EL Builder.

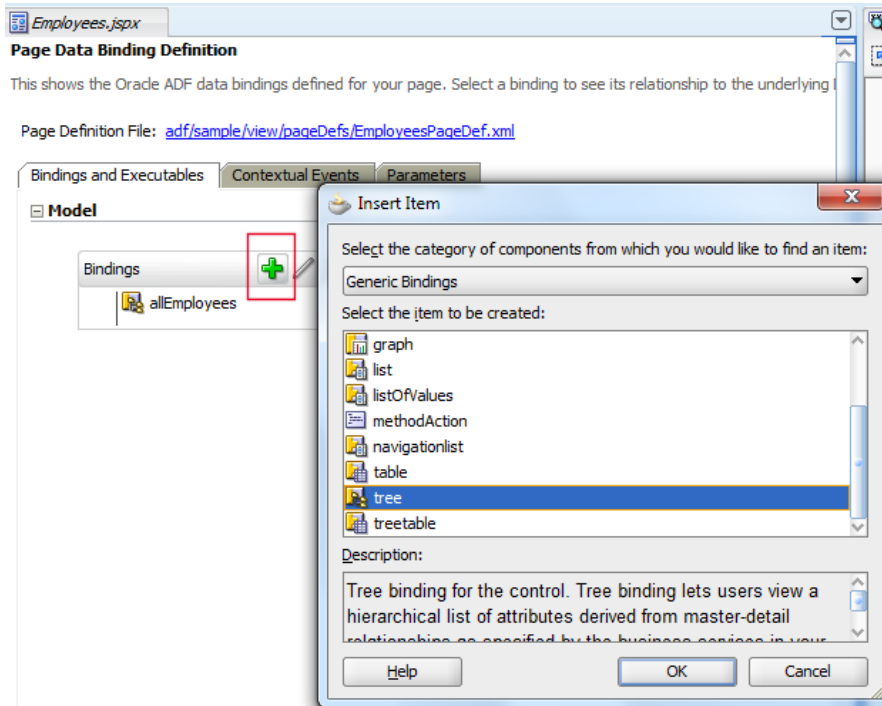


Expand the “filterCriteria” node and select the attribute name of the attribute that is represented by the table column to filter. In the example below, we are going to show a list of departments to help filtering the entries for a specific department.

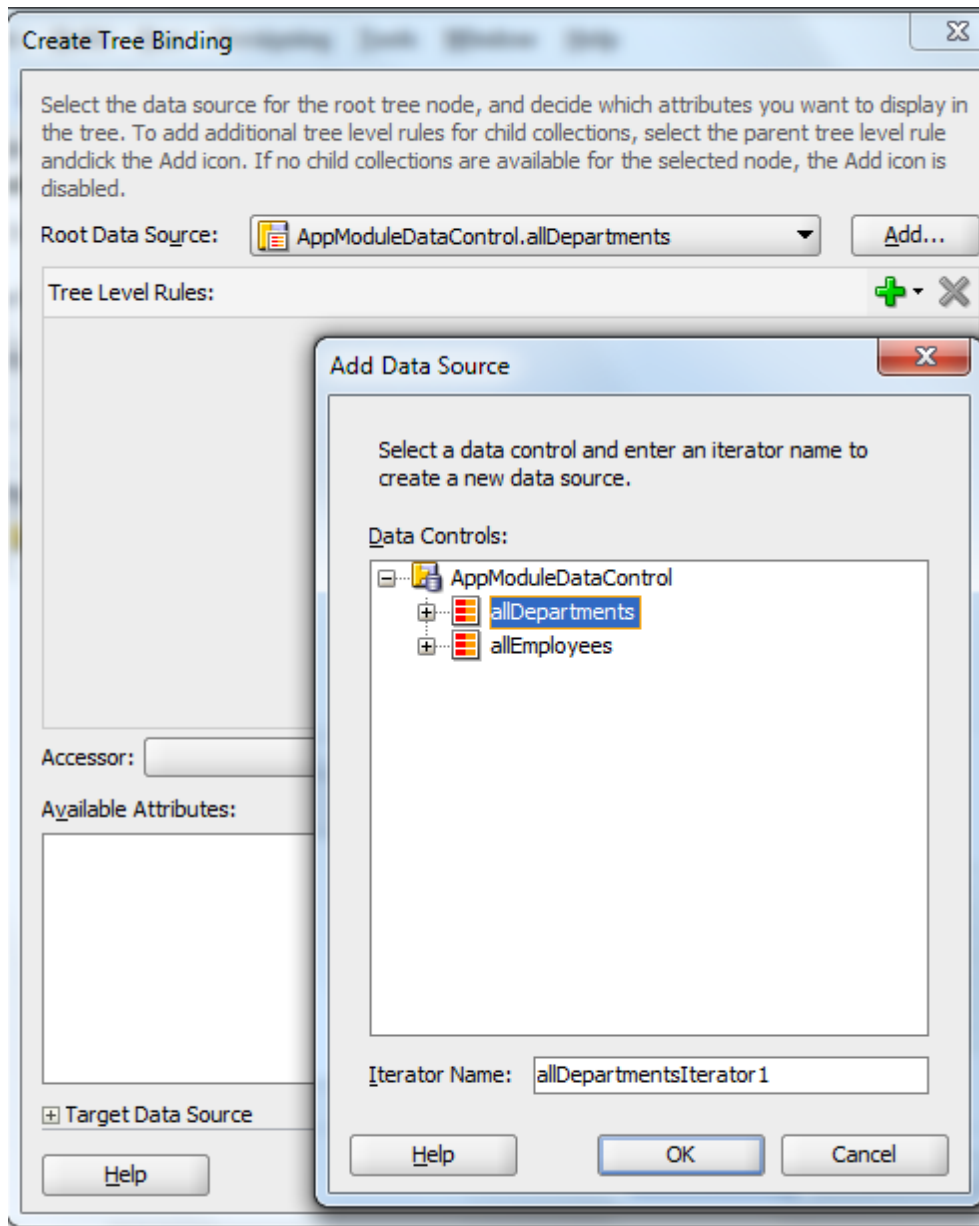
Having done this, the table now shows a selectManyChoice component configured to set the table filter for the DepartmentId attribute. Next is to create the list of values.



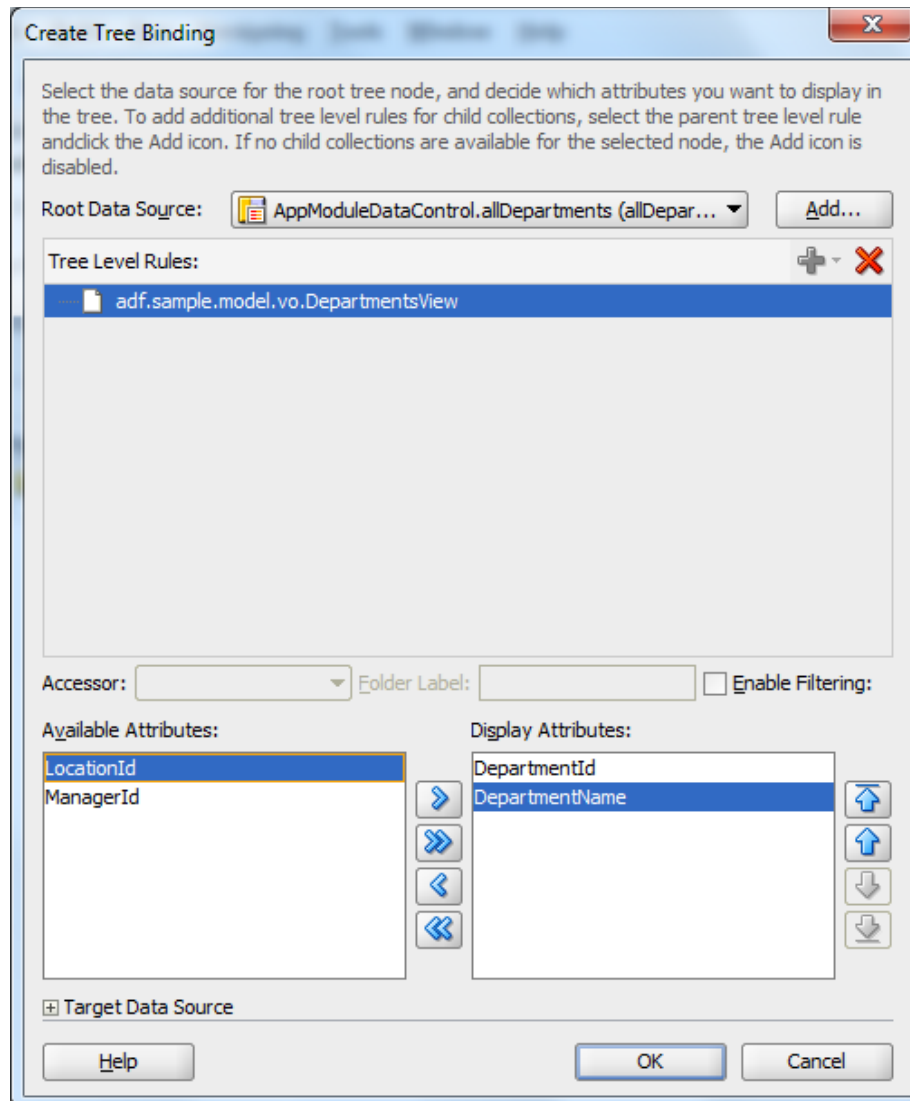
In the example, instead of using a static list of values, we read the department names from an ADF binding so the list is produced dynamically.



To create the ADF binding for the list, select the “Bindings” tab at the bottom of the JSF page. In the binding editor, press the green plus icon in the “Bindings” section and choose “tree” from the list of “Generic Bindings”. The tree binding can be bound to a collection – like Departments View Object in the example and return a list of values to display in the `af:selectManyChoice` component.



Create a single level tree, as shown in the image above. Press the “Add” button to select a collection, which then implicitly creates an iterator, and then press the green plus icon to create the tree rule.



Form the available list of attributes, select at least one. In the example above, the plan is to show the department name to the user and internally use the department Id when filtering the table.

Note: When the binding is created, navigate to the PageDef file and select the iterator that was created for the tree binding. The iterator is in the “Executable” section of the PageDef file. Select the iterator and open the Property Inspector. Change the “RangeSize” property from “10”, the default value to “-1”, so the list shows all available list values

Note: Be sensible with how many list values you show in the selectManyChoice. If, for example, the departments table had 1 0000 0000 rows then an af:selectManyChoice as the table filtering component is not a good choice. In this case you would use a LOV.

To populate the af:selectManyChoice components with values read from the ADF binding, we use an af:forEach component that iteratively created instance of f:selectItem components for each list entry. For this, you remove the f:selectItems entry that is created when adding the af:selectManyChoice component.

The page source should look as shown below


```

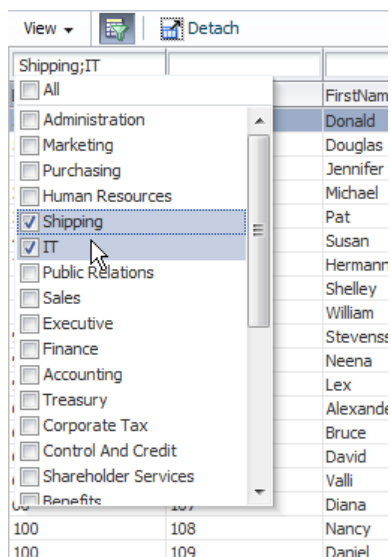
<af:column sortProperty="DepartmentId" filterable="true"
    sortable="true" id="c11"
    headerText="#{bindings.allEmployees.hints.DepartmentId.label}">
<af:outputText value="#{row.DepartmentId}" id="ot5">
    <af:convertNumber groupingUsed="false"
        pattern="#{bindings.allEmployees.hints.DepartmentId.format}"/>
</af:outputText>
<f:facet name="filter">
    <af:selectManyChoice id="soc1"
        value="#{vs.filterCriteria.DepartmentId}">
        <af:forEach var="listrow"
            items="#{bindings.allDepartments.rangeSet}">
            <f:selectItem id="si1"
                itemValue="#{listrow.DepartmentId}"
                itemLabel="#{listrow.DepartmentName}"/>
            </af:forEach>
        </af:selectManyChoice>
    </f:facet>
</af:column>

```

The `af:forEach` component references the ADF tree binding, `#{bindings.allDepartments.rangeSet}`, which queries the Departments View Object for the available department Ids and names.

The `af:forEach` component has a “`var`” attribute value defined as “`listrow`”, which is used to populate the select item label and value.

At runtime the table shows as in the image below. The `af:selectManyChoice` displays all departments for the user to choose from.



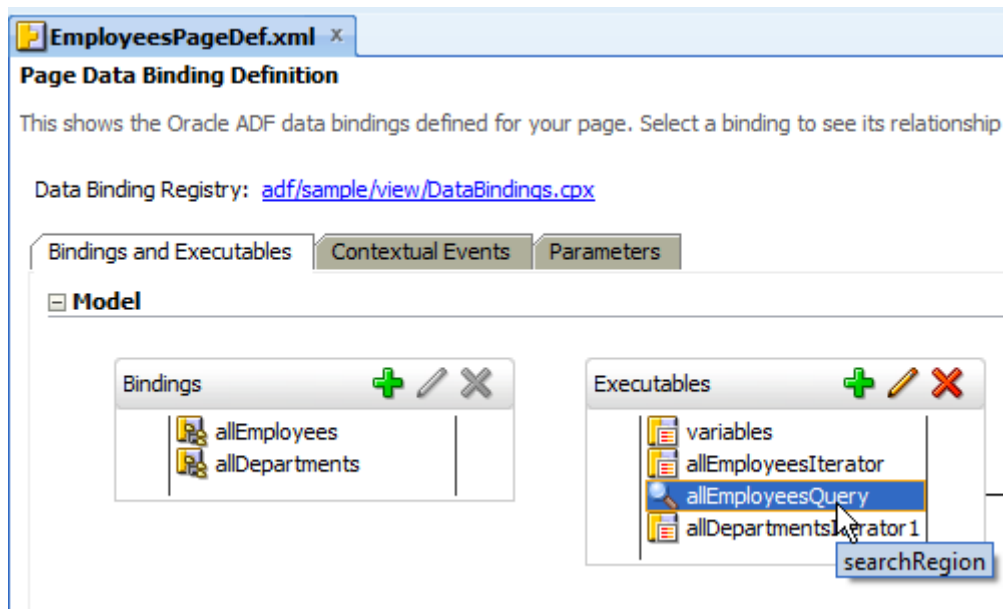
However, unlike the Select one choice solution in sample #16, the select many choice **is not yet fully functional** and additional configuration is required mainly to translate the list of selected values into a string.

Making the `af:selectManyChoice` filter work

The `af:selectOneChoice` component in ADF Code Corner sample #16 returns a single criteria value to filter the table. The `af:selectManyChoice` component however returns a `java.util.List` of values that needs to be translated into a format understood by the search binding executing filtered query. To implement the solution, two things need to happen

- The table's **QueryListener** property needs to be overwritten to execute custom logic that preserves the default query behavior, but that has a change to manipulate the query criteria
- A managed bean needs to be created that holds the custom query listener and that translates the select many list values to a String format, executes the query and then sets the filter back to the `List` value to restore the user filter selection

Using Oracle ADF bound filterable ADF Faces tables, the table's `QueryListener` property is configured to point to a search binding that automatically gets created in the Executable section of the PageDef file.

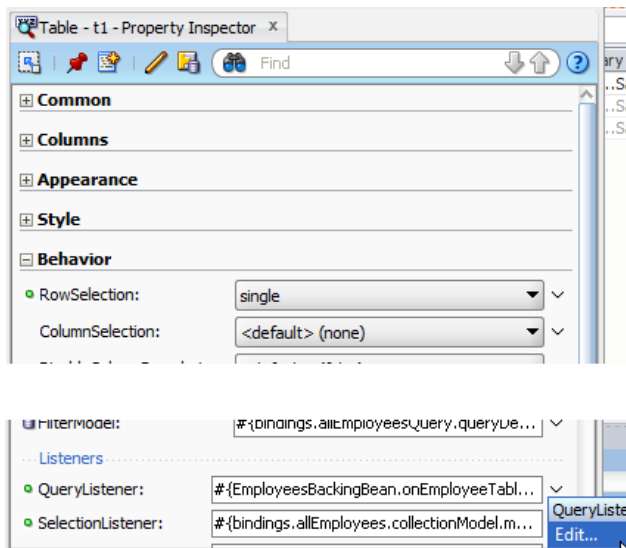


The search binding is of type `FacesCtrlSearchBinding` (extending `JUFormBinding` and `DCBindingContainer`) and exposes a method **processQuery** to perform the filtered query. The EL expression on the `QueryListener` property looks similar to the one shown below:

```
#{bindings.allEmployeesQuery.processQuery}
```

To handle the `af:selectManyChoice` component in the table filter, the `QueryListener` property needs to be changed so it points to a managed bean handling the query event. However, the default ADF behavior when querying the table should be preserved and therefore the original EL should be copied to the clipboard first.

Use the "arrow icon" next to the **QueryListener** property to display the context menu and press the **Edit** option. Create or select a managed bean and define a name for the **QueryListener** handler method. At the end, the **QueryListener** property should be configured similar as shown in the image below.



The managed bean query listener used in the sample is shown below. The code is commented for you to understand what it does:

```
import java.util.ArrayList;
import java.util.Map;
import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.faces.application.Application;
import javax.faces.context.FacesContext;
import oracle.adf.view.rich.event.QueryEvent;
import oracle.adf.view.rich.model.FilterableQueryDescriptor;
import oracle.jbo.domain.Number;

public class EmployeesBackingBean {
    public EmployeesBackingBean() {
    }

    //Custom Query Listener to apply af:selectMany choice values to
    //the table filter criteria
    public void onEmployeeTableQuery(QueryEvent queryEvent) {
        //user selected values
        ArrayList<Object> departmentIdArray = null;
        FilterableQueryDescriptor fqd =
            (FilterableQueryDescriptor) queryEvent.getDescriptor();

        //current filter criteria as submitted by the user
        Map _criteriaMap = fqd.getFilterCriteria();
        //Translate DepartmentId array list to OR separate list of values
        //for the filter to work.
        StringBuffer deptIdFilterString = new StringBuffer();
```

```
if (_criteriaMap.get("DepartmentId")!=null){
    departmentIdArray =
        (ArrayList<Object> )_criteriaMap.get("DepartmentId");

    for (int argIndex = 0; argIndex < departmentIdArray.size();
        argIndex++) {

        //You need to know what is the underlying data type you are
        //dealing with for the attribute. If you are on 11gR1(11.1.1.x)
        //then this type is jbo.domain.Number for numeric attributes.

        //
        //If you are on 11g R2 (11.1.2.x) this could be
        //oracle.jbo.domain.Number,
        //Integer or BigDecimal. If you use 11g R2, check the View
        //Object for the attribute data type
        if(argIndex==0){
            //first argument has no "OR"

            //this sample used oracle.jbo.domain.Number for the
            //DepartmentId attribute

            Number departmentId =
                (Number) departmentIdArray.get(argIndex);
            deptIdFilterString.append(departmentId.toString());

        }
        else{
            //any subsequent argument is OR'ed together
            deptIdFilterString.append(" OR ");
            Number departmentId = (Number)
                departmentIdArray.get(argIndex);
            deptIdFilterString.append(departmentId.toString());
        }
    }

    //for some reasons, if in a single value select case, the
    //filter breaks and an error message is printed that the
    //String representation of the single value isn't found in
    //the list. The line below fixes the problem for filter values
    //that are positive numbers
    deptIdFilterString.append(" OR -1");

    String departmentIds = deptIdFilterString.toString();
    _criteriaMap.put("DepartmentId", departmentIds);
    fqd.setFilterCriteria(_criteriaMap);
}

// preserve default query listener behavior
//#{bindings.allEmployeesQuery.processQuery}
```

```

FacesContext fctx = FacesContext.getCurrentInstance();
Application application = fctx.getApplication();
ExpressionFactory expressionFactory =
    application.getExpressionFactory();
ELContext elctx = fctx.getELContext();
MethodExpression methodExpression =
    expressionFactory.createMethodExpression(
        elctx,
        "#{bindings.allEmployeesQuery.processQuery}",
        Object.class,
        new Class[] {QueryEvent.class});
methodExpression.invoke(elctx, new Object[] {queryEvent});

//restore filter selection done by the user. Note that this
//needs to be saved as an ArrayList
_criteriaMap.put("DepartmentId", departmentIdArray);
fqd.setFilterCriteria(_criteriaMap);
}
}

```

Download the sample

You can download the Oracle JDeveloper 11.1.1.6 sample workspace as example #98 from the ADF Code Corner website

<http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html#CodeCornerSamples>

Configure the database connect (View → Database → Database Navigator) to connect to the HR schema of your Oracle XE, standard or enterprise database. Then run the JSPX page to try the select many filter. When you select values for the **DepartmentId**, click outside of the select list before pressing the enter key.

This solution works with any Oracle JDeveloper 11g R1 and R2 version but was tested with 11.1.1.6 only.

RELATED DOCUMENTATION

☒	ADF Code Corner #16 " How-to customize the ADF Faces Table Filter" http://www.oracle.com/technetwork/developer-tools/adf/learnmore/16-custom-table-filter-169145.pdf
☒	
☒	